Using Graphs to Analyze Big Linked Data

Hassan Chafi,
Director, Research and Advanced Development
Oracle Labs

ORACLE

Safe Harbor Statement

The following is intended to outline our research activities and general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Big Data and Data Analysis

- The Big Data era is here
  - Volume
  - Velocity
  - Variety
- However, just storing and managing this data is not sufficient
  - Typically Big Data is low value per byte

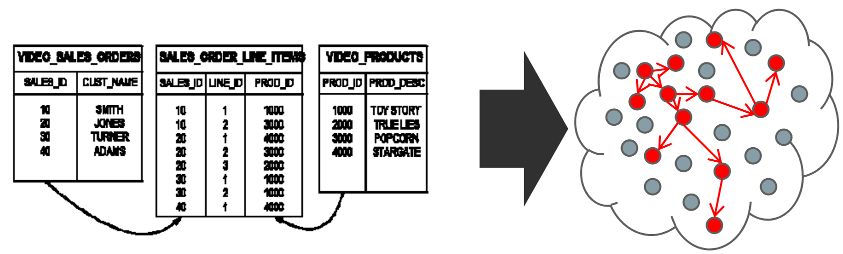
➔ We want to get useful information out of the huge data sets

Data Analytics:
- Classic OLAP/BI
- Statistical analysis
- Machine learning
- Graph analysis

ORACLE

Graph Analysis

- Represent your data as a graph
  - Data entities become vertices
  - Relationships become edges



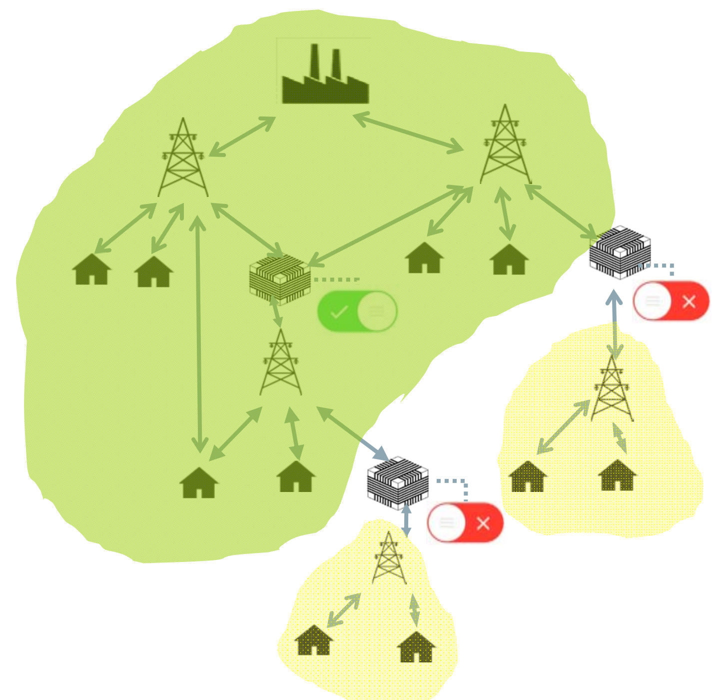
- Analyzing the graph can yield very useful information
  - As it inspects fine-grained relationships between data entities and can factor in the shape of the resulting graph
- Challenge:
  - Traditional tools are neither convenient nor efficient for graph analysis

➔ Let's start with a very simple real world example

ORACLE

Example #1: Power Distribution Network

- Real world use case from an infrastructure company
- [The data set]
  - The dataset represents a power distribution network
  - Generators, transformers, lines, regulators, switches, …
- [One simple question] (out of many)
  - What/how many elements still remain reachable from a source, when given switches are turned off



Example #1: Relational Data Model

- Their data was originally stored in relational tables*

ID	Name	Type	Voltage…
2018281	XFM_Sub	Generator	…
27080172	SW_35	Switch	…
…			

Table for node information

Connection	Node A	Node B	Node C	Node D	Node E…
6693	2018281	289301	4985701	-	-
7207	2019182	495812	9191913	4985701	-
…	…	…	…		

Table for connection information (using null values for variable connection size)

Issue #1
The question cannot be answered with simple SQL
➔ Cannot avoid some programming (e.g. PL/SQL)

Issue #2
Performance issues (next slide)

*Schema simplified for explanation's sake

ORACLE

Example #1: Performance Issue

- What needs to be done:

Overall, many joins during each of many iterations!

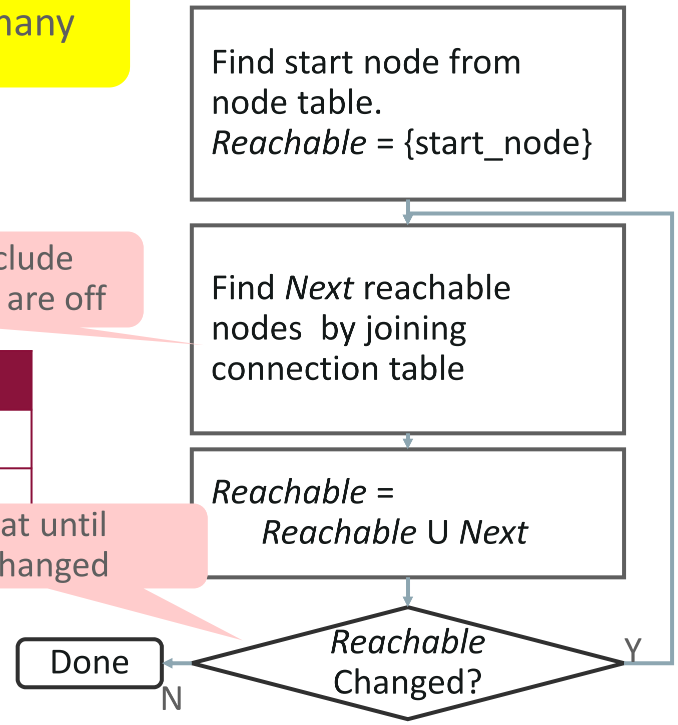
ID	Name	Type	Voltage...
2018281	XFM_Sub	Generator	...
27080172	SW_35	Switch	...
...			

Need to exclude switches that are off

Connection	Node A	Node B	Node C	Node D	Node E...
6693	2018281	289301	4985701	-	-
7207	2019182	495812	9191913	4985701	-
...	...	...	...	...	

289301

Repeat until not changed

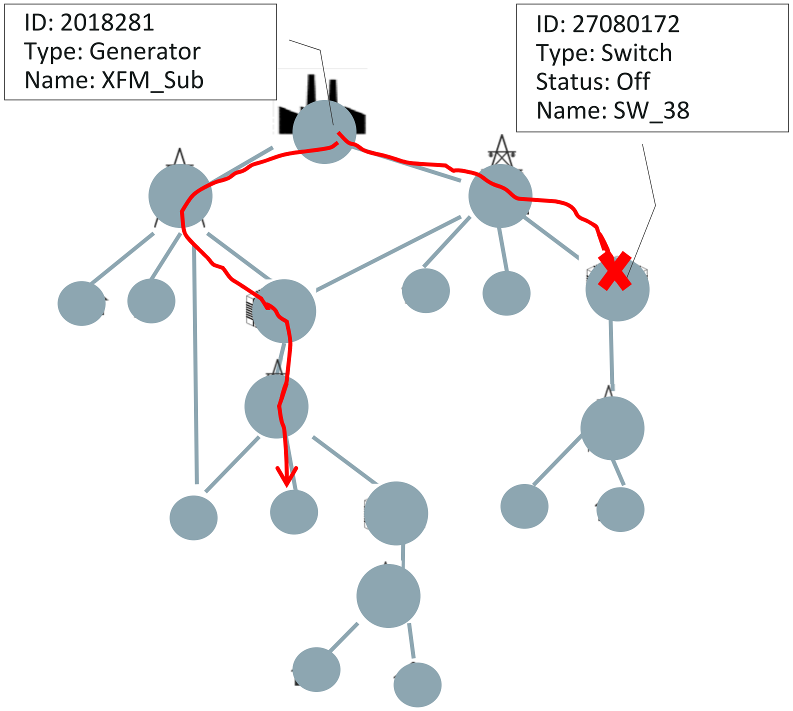


ORACLE

Example #1: Use A Graph Model, Instead

- Represent the data as a graph
  - Vertices and edges have extra information, or *properties*
  - Fits very naturally
- Answer the question in natural ways
  - Starting from the given vertex,
  - traverse the graph,
  - without going through ‘off’ switches

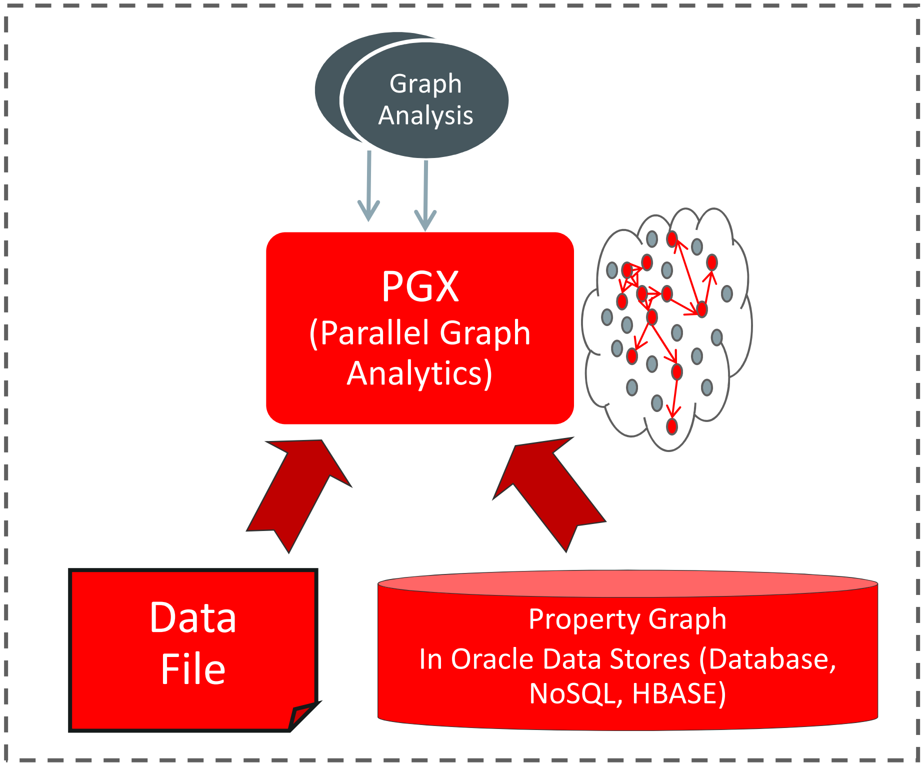
But, what tools provide such data model and operations?



ORACLE

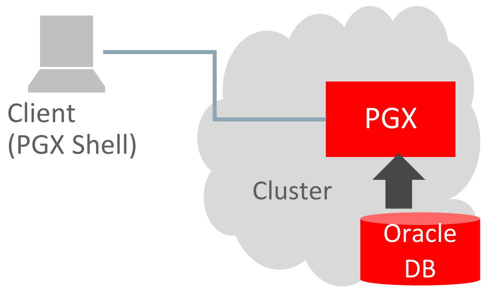
Introducing PGX

- PGX is a graph processing system from Oracle Labs
- PGX reads graph data from files or Oracle DBs
- PGX builds and maintains its own graph representation
- … and provides graph analysis features to the users



ORACLE

Example #1: PGX Example



- PGX supports server-client mode execution
- Multiple concurrent clients
- *Groovy*-(and iPython) based interactive shell

Load graph and get root vertex

```
pgx> G = session.readGraphWithProperties(“ … ”)
pgx> root = G.getVertex(21474836490)
```

Create a filter condition.
Do BFS(Breadth-First Search) traversal from root

```
pgx> navigator = new VertexFilter(
  “vertex.type!=‘switch’ or vertex.state==true“)
pgx> analyst.filteredBfs( G, root, null, navigator )
```

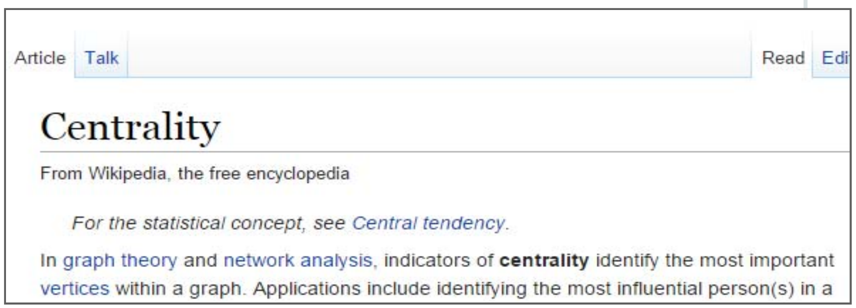
Find reached nodes, and print their distances from the root

```
pgx> G.queryPgql(
  “SELECT vertex, vertex.distance   \
   WHERE (vertex WITH distance >=0) \
   ORDER BY vertex.distance DESC”).print(30)
```

ORACLE

Example #2: Influencer Identification

- General Idea
  - Capture relationships between data entities as a graph
  - Inspect the topology of the graph and identify *important* entities
- More Theoretic Approach
  - Centrality in Graph theory

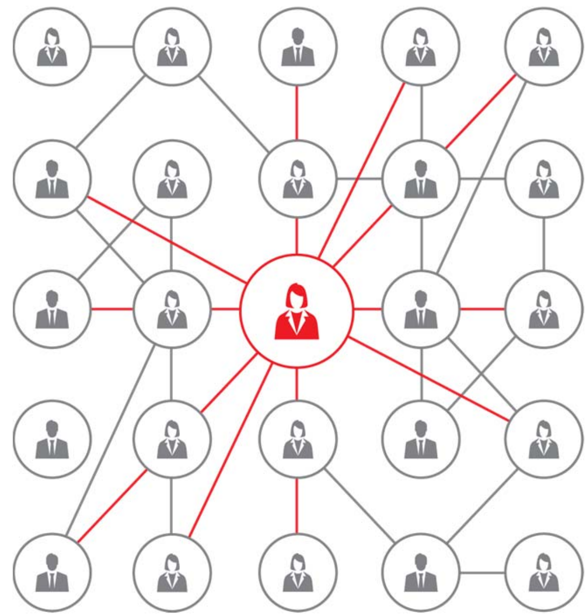

Article Talk Read Edi

Centrality

From Wikipedia, the free encyclopedia

For the statistical concept, see Central tendency.

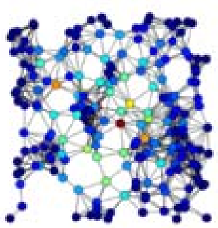
In graph theory and network analysis, indicators of **centrality** identify the most important vertices within a graph. Applications include identifying the most influential person(s) in a

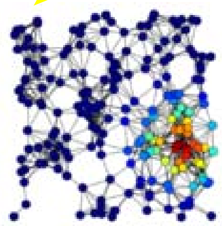


Example #2: Centrality Algorithms?

- Graph theory defines many different measures
  - Betweenness Centrality
  - Closeness Centrality
  - Eigenvector Centrality
  - Pagerank
  - HITS
  - …

Each algorithm suggests different definition of *importance*

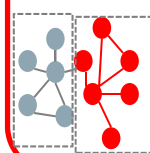

Betweenness centrality


Eigenvector centrality

(images from Wikipedia)

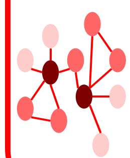
- PGX comes with built-in algorithms – various centralities included

Detecting Components and Communities



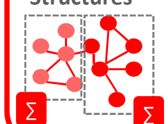
Tarjan's, Kosaraju's, Weakly Connected Components, Label Propagation (w/ variants), Soman and Narang's Spacification

Ranking and Walking



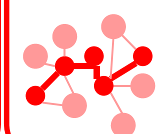
Pagerank, Personalized Pagerank, Betwenness Centrality (w/ variants), Closeness Centrality, Degree Centrality, Eigenvector Centrality, HITS, Random walking and sampling (w/ variants)

Evaluating Community Structures



Conductance, Modularity, Clustering Coefficient (Triangle Counting), Adamic-Adar

Path-Finding



Hop-Distance (BFS), Dijkstra's, Bi-directional Dijkstra's, Bellman-Ford's

Link Prediction SALSA (Twitter's Who-to-follow)

Other Classics Vertex Cover, Minimum Spanning-Tree (Prim's)

ORACLE

Example #2: Let’s try with a *real* data set

- A network of characters in Marvel comics
- http://exposedata.com/marvel/

1. Download .csv file; it is a simple list of *linked* characters

```
"NOVA/RICHARD RIDER","HULK/DR. ROBERT BRUC"
"NOVA/RICHARD RIDER","NIGHT THRASHER/DUANE"
"NOVA/RICHARD RIDER","SPIDER-MAN/PETER PAR"
"SPIDER-MAN/PETER PAR","FIRESTAR/ANGELICA JO"
```

2. Write up a simple JSON descriptor

```
{
 “uri”: “hero-network.csv”,
 “format”: “edge_list”,
 “node_id_type”: “string”,
 “separator”: “,”
}
```

We want the graph *undirected*

3. Start PGX and load the graph

```
pgx> G = session.readGraphWithProperties(“hero-network.csv.json”).undirect();
```

ORACLE

Example #2: Running Centrality Algorithms

```
pgx> analyst.pagerank(G, 0.0001, 0.85, 100)
pgx> analyst.vertexBetweenessCentrality(G)


pgx> G.queryPGQL(“\
   SELECT n, n.pagerank, n.betweenness \
   WHERE (n) ORDER BY DESC (n.pagerank)”).print(10);


pgx> G.queryPGQL(“\
   SELECT n, n.pagerank, n.betweenness \
   WHERE (n) ORDER BY DESC (n.betweenness)”).print(10);
```

Compute pagerank and betweenness centrality

Query the graph and get top 10 pagerank and betwenness centrality

Want to see the results? (next slide)

ORACLE

All images are from Wikipedia

Example #2: And the Award goes to …

- Top-10: Pagerank

1. Captain America
2. Spider-Man
3. Iron Man
4. Wolverine
5. Thing
6. Thor
7. Human Torch
8. Mr. Fantastic
9. Scarlet Witch
10. Invisible Woman

```
THING/BENJAMIN J. GR   0.006324107190160558

        n.betweenness
=====================
3720705.946340419
5033505.395979824
2061886.226671595
2055461.3906177823
1355105.9963565439
1464283.2128501285
1094332.8826185146
1133204.6525742295
829059.8688294408
723990.9267695188
```

There are correlations between two measures

Inner circle characters tend to get higher Pagerank (Avengers, Fantastic Four)

- Top-10: Betwenness Centrality

1. Spider-Man
2. Captain America
3. Iron Man
4. Wolverine
5. Dr. Strange
6. Havok
7. Thor
8. Hulk
9. Thing
10. Daredevil

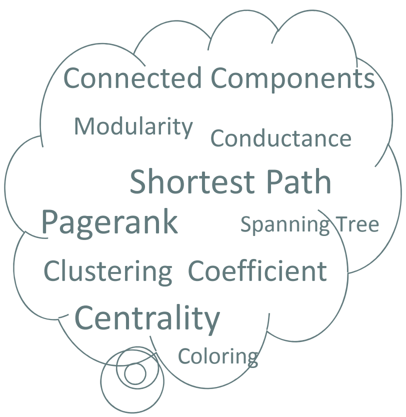
```
        n.betweenness
=====================
5033505.395979824
3720705.946340419
2061886.226671595
055461.3906177823
1628817.0459632503
1612813.8824396846
464283.2128501285
```

Links to various X-men characters including minor ones

Characters that link to a different character group (e.g. their own enemies)

Two Kinds of Graph Workloads

Computational Graph Analytics



Compute certain values on nodes and edges

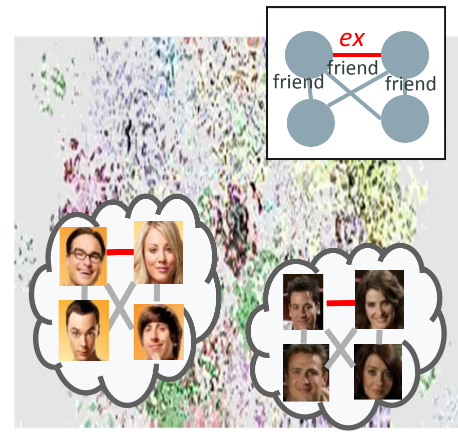
While (repeatedly) **traversing** or **iterating** on the graph

In certain **procedural** ways

Graph Pattern Matching

Given a **description** of a pattern

Find every sub-graph that **matches** it



PGX supports **both** kinds, as well as **combinations of the two**

Images from IMDB.com

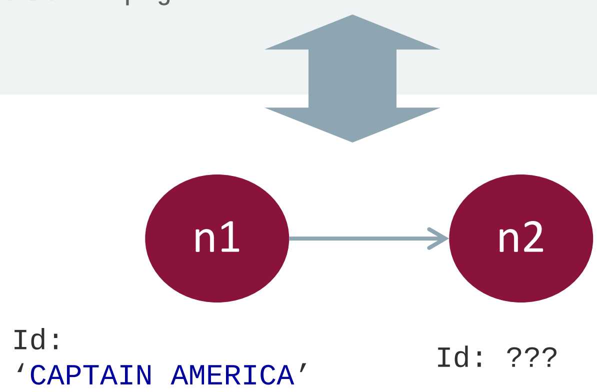
Graph Pattern Matching

- PGQL
  - A query language for graph pattern match
  - SQL-like syntax but with graph pattern description and property access

Example> Retrieve all the neighbors of Captain America, list them ordered by their Pagerank values

```
SELECT n2
WHERE
   (n1 WITH id() = 'CAPTAIN AMERICA') -> n2
ORDER BY n2.pagerank
```

- SQL Like syntax format:
  `SELECT, WHERE, ORDER BY`
- Graph pattern description:
  `(n1 … ) -> n2`
- Property access :
  `(n1 WITH id() = '…' )`
  `n2.pagerank`



Images from Wikipedia, marvel.wikia.com

Example #2: More Information with Patterns

- PGQL enables you to ask more interesting queries

```
pgx> G = G.simplify( … )

pgx> G.queryPgql(“SELECT x \
   WHERE (a WITH id()=‘SHANG-CHI’)->x,\
         (b @ ‘WHITE TIGER/HECTOR A’)->x,
         (c @ ‘IRON FIRST/DANIEL RAN’)->x\
”).print(20);

pgx> G.queryPgql(“SELECT b, x  \
   WHERE (a@‘DR. OCTOPUS/OTTO OCT’)->x<-b,\
         x.pagerank < 0.0001 ,\
         b.pagerank > 0.005 \
”).print(10);
```

Remove multiple edges to avoid repeated answers

Who are the common neighbors of three martial-art heroes:

Short-cut syntax for Id() matching

Find Dr. Ock’s *minor* neighbors which has link to a *major* character (assumed by pagerank value)

Computational analysis combined with pattern matching

IRON MAN
SPIDER-MAN
MOON KNIGHT
FU, MANCHU
… (21 results)

SPIDER-MAN
CHANCE II
HUMAN TORCH
GAYLE WATSON
… (100 results)

ORACLE

What if I need other algorithms?

- Graph algorithms are evolving
  - e.g., Scientists keep proposing different definitions of centrality
- What if I want a custom algorithm than built-in one?
- In other words, how programmable is PGX?

- In two ways:

(1) Via Java API

- `getVertex(), getProperty(), …`
- allows fine-grained control
- the user implements his/her algorithm

(2) **Via a Domain-Specific Language (DSL)**

- intuitive programming
- automated parallelization and optimization

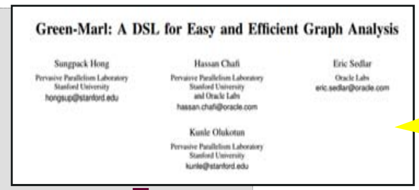
(more in next slide)

ORACLE

PGX -- Flexibility through DSL

ASPLOS 2012

Custom algorithms can be implemented with Green-Marl (DSL)



Green-Marl: A DSL for Easy and Efficient Graph Analysis

Sungpack Hong, Pervasive Parallelism Laboratory, Stanford University

Hassan Chafi, Pervasive Parallelism Laboratory, Stanford University and Oracle Labs, hassan.chafi@oracle.com

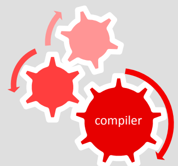
Eric Sedlar, Oracle Labs, eric.sedlar@oracle.com

Kunle Olukotun, Pervasive Parallelism Laboratory, Stanford University

Green-Marl is a DSL designed for graph analysis:

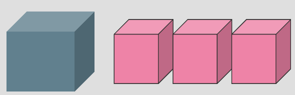
- Intuitive, imperative language suitable for describing algorithms
- provides first class support for graph entities – node, edge, …

PGX compiler applies **parallelization** and **optimization**

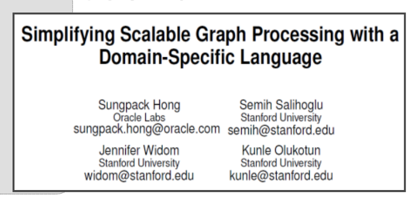


Powerful optimizations are enabled by high-level semantic knowledge

… **compiles** and runs on PGX single node (or cluster)



CGO 2014



Simplifying Scalable Graph Processing with a Domain-Specific Language

Sungpack Hong, Oracle Labs, sungpack.hong@oracle.com

Semih Salihoglu, Stanford University, semih@stanford.edu

Jennifer Widom, Stanford University, widom@stanford.edu

Kunle Olukotun, Stanford University, kunle@stanford.edu

DSL Example: Betweenness Centrality

```
procedure my_bc(G: graph;        // input graph
      BC: nodePropery<double>)  // output node property
{
  G.BC = 0; // Initialize output

  foreach (s:G.nodes) { // from each node s in G

    // create temporary node properties and initialize them
    nodeProperty<double> sigma;
    nodeProperty<double> delta;
    G.sigma = 0;
    s.sigma = 1;

    // Do BFS traversal from s
    inBFS(v: G.nodes from s) (v != s) {
      // compute sigma for v, by summing over BFS parents
      v.sigma = sum(w:v.upNbrs) { w.sigma };
    }
    inReverse(v!=s) { // Do reverse-BFS order iteration to s
      v.delta =       // compute delta for v, by summing over BFS children
        sum (w:v.downNbrs){(1+ w.delta)/w.sigma} * v.sigma;

      v.BC += v.delta ; // accumulate delta into BC
    }
  }
}
```

Green-Marl Program

Writing graph algorithm becomes straight-forward. The language provides graph-specific intrinsics such as graph, property, BFS, neighbors ..

Our intelligent compiler sees parallelizable operations in the algorithm, and (selectively) applies them automatically.

ORACLE

DSL Example: Betweenness Centrality

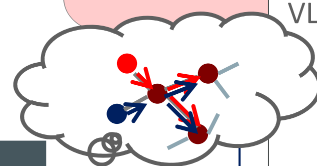
```
procedure my_bc(G: graph;        // input graph
      BC: nodePropery<double>)  // output node property
{
  G.BC = 0; // Initialize output

  foreach (s:G.nodes) { // from each nodes s in G

    // create temporary node properties and initialize them
    nodeProperty<double> sigma;
    nodeProperty<double> delta;
    G.sigma = 0;
    s.sigma = 1;

    // Do BFS traversal from s
    inBFS(v: G.nodes from s)(v != s) {
      // compute sigma for v, by summing over BFS parents
      v.sigma = sum(w:v.upNbrs) { w.sigma };
}
```

Green-Marl Program

The compiler can apply even more powerful optimizations.

For instance, the compiler sees that BFS is repeated inside a parallel loop

Then the compiler merges multiple BFS traversals into one



VLDB 2015

The More the Merrier: Efficient Multi-Source Graph Traversal

Manuel Then* then@in.tum.de
Moritz Kaufmann* kaufmanm@in.tum.de
Fernando Chirigati† fchirigati@nyu.edu
Tuan-Anh Hoang-Vu† tuananh@nyu.edu
Kien Pham† kien.pham@nyu.edu
Alfons Kemper* kemper@in.tum.de
Thomas Neumann* neumann@in.tum.de
Huy T. Vo† huy.vo@nyu.edu
* Technische Universität München
† New York University

	PGX	iGraph (C++ library)
Betwenness Centrality	80 mins	Not finished after 48 hours
Closeness Centrality	9 mins	Not finished after 48 hours

Performance is greatly improved !
(measured with 200 million edged graph)

ORACLE

How does it work with in PGX?

Compile Custom algorithm

```
pgx> my_bc = session.compile("my_bc.gm")
```

Read a graph and run the algorithm

```
pgx> G = session.readGraphWithProperties(" … ")
pgx> P = G.createVertexProperty(PropertyType.DOUBLE, "my_prop");
pgx> my_bc.run(G, P);
```

Creating a new vertex property to hold the output

Compiled methods can be naturally invoked from Groovy (or Java)

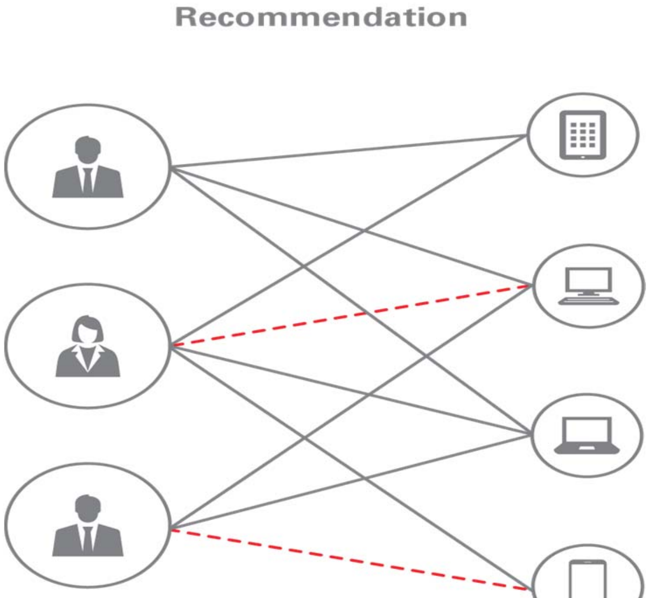
Compare to Built-in Implementation

```
pgx> anayst.vertexBetweennessCentrality(G)
pgx> G.queryPgql(
  "SELECT n, n.my_prop, n.betweenness   \
   WHERE (n) \
   ORDER BY DESC(n.betweenness").print(10);
```

n	n.my_prop	n.betweenness
SPIDER-MAN/PETER PAR	5033505.395979839	5033505.395979841
CAPTAIN AMERICA	3720705.946340431	3720705.9463404035
IRON MAN/TONY STARK	2061886.2266715805	2061886.2266715826
WOLVERINE/LOGAN	2055461.3906177846	2055461.3906177809
DR. STRANGE/STEPHEN	1628817.0459632506	1628817.0459632422
HAVOK/ALEX SUMMERS	1612813.882439691	1612813.8824396855

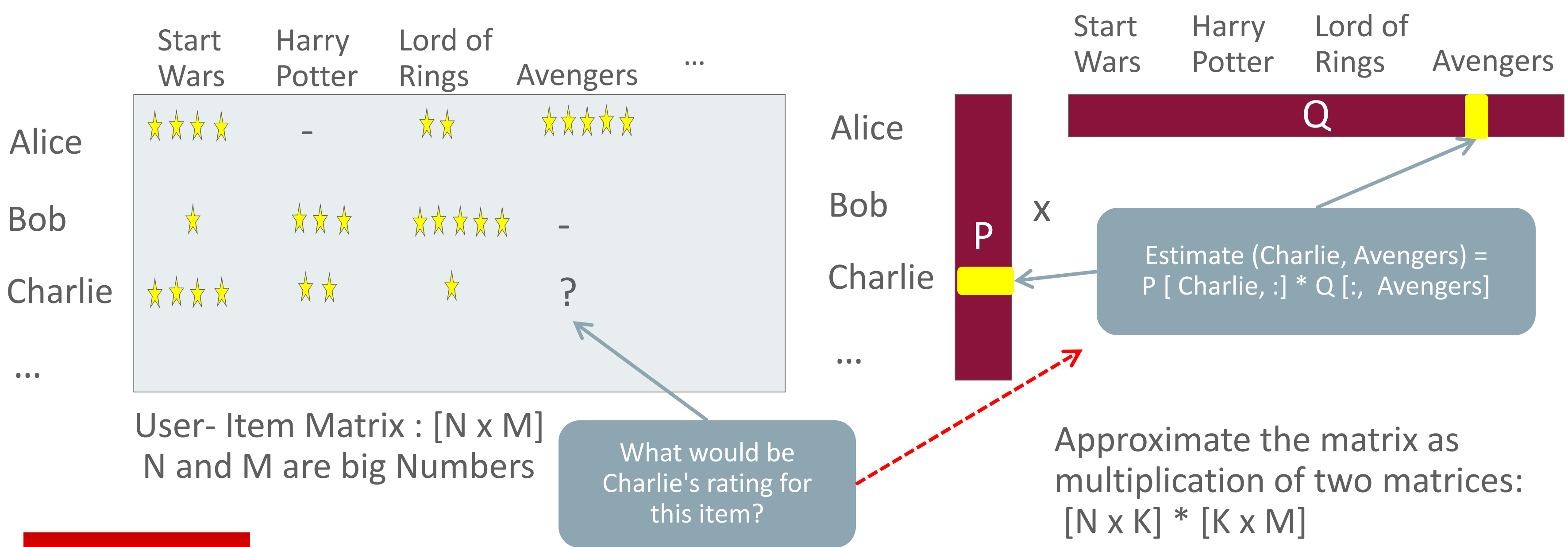
Example #3: Recommendation

- Data set and Question
  - Users and Items
  - Items purchases by each user and their ratings
- [Q] Given a user, can we suggest an item that he/she would like to buy next?
- General Idea: Collaborative Filtering
  - Find users that are *similar* to the given user
  - Find items that are favored by those users
  - Recommend such items



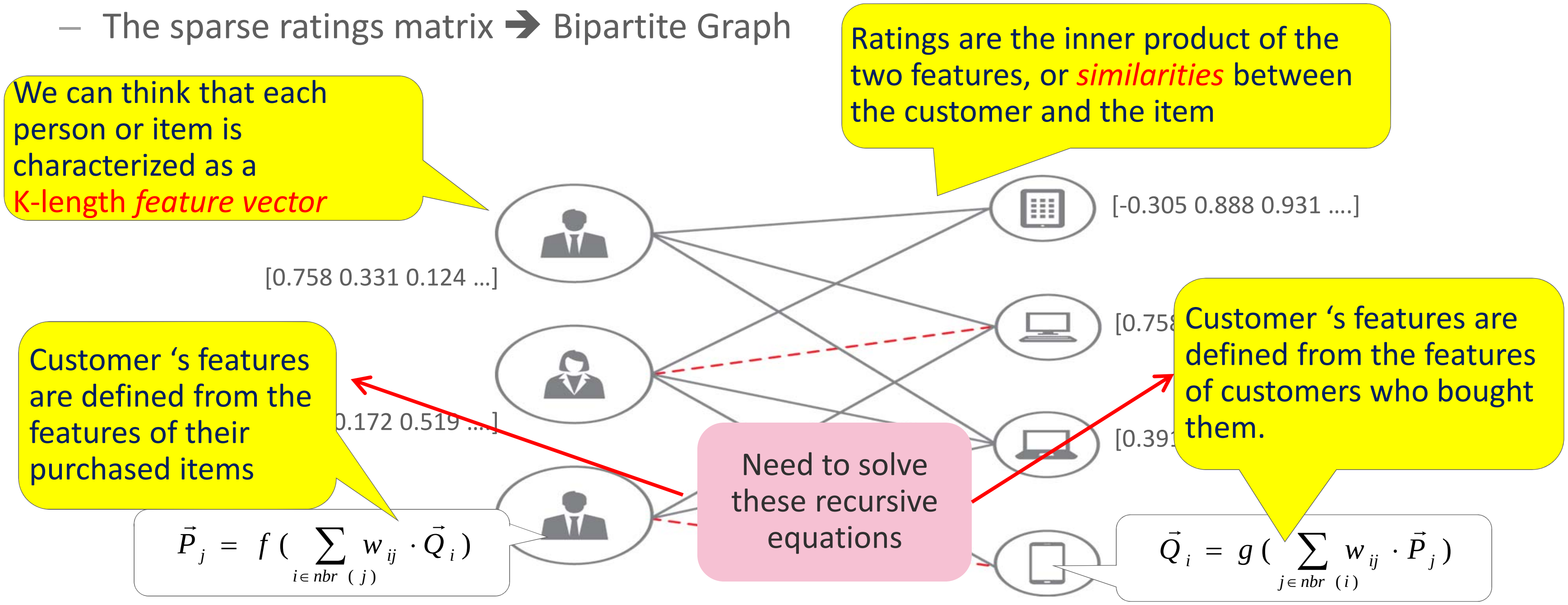
Example #3: Mathematical Backgrounds

- Matrix Factorization
  - Consider the ratings data as a (sparse) matrix



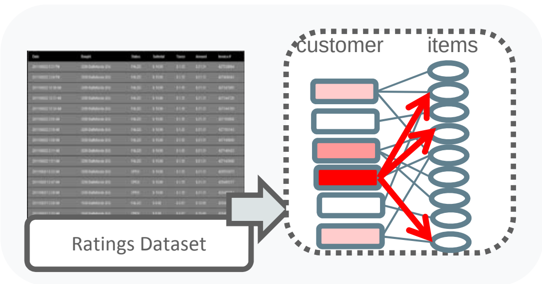
Example #3: Graph Intuition

- The exactly same technique can be applied with graph representation
  - The sparse ratings matrix ➔ Bipartite Graph



Example #3: PGX Application Flow

1. Data Acquisition
   - Ratings dataset are loaded as a bipartite graph



2. Training
   - Compute feature vectors
   - PGX provides matrix factorization implementation

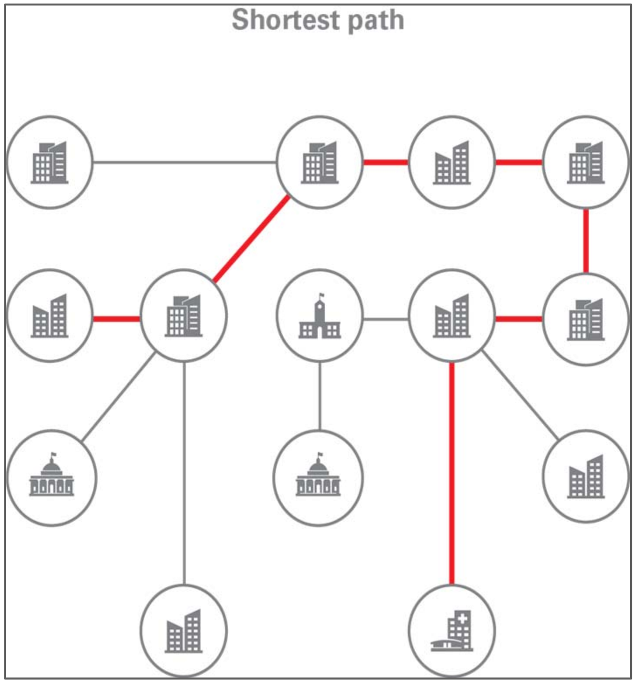
```
while (i < max_step) {
    double rmse = 0.0;

    foreach (u: G.nodes) {
        if (u.is_left) {
            vect<double>[K] Z = 0.0;

            for(e: u.outEdges) {
                node v = e.toNode();

                double w1 = e.weight;
                double w2 = u.PQ * v.PQ;

                if (w2 > VALUE_MAX) {
                    w2 = VALUE_MAX;
                } else if (w2 < VALUE_MIN) {
                    w2 = VALUE_MIN;
                }

                Z += ((w1 - w2) * v.PQ - lambda *
                rmse += (w1 - w2) * (w1 - w2);
            }
```

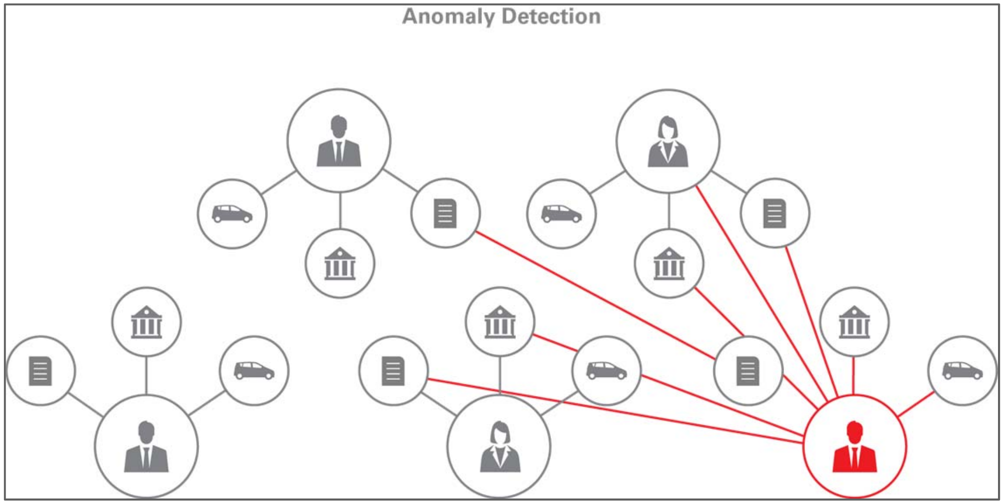
3. Recommend
   - Given a user and set of items, compute inner products of feature vectors.
   ➔ Recommend Top-K items

Vermont Is For Lovers - 5.920148497542082
Night Flier - 5.691110431206573
Boys Life - 5.682416396766282
Frisk - 5.654641389008914
Visitors, The (Visiteurs, Les) - 5.579409110367704
Jerky Boys, The - 5.546784101675797
Quartier Mozart - 5.531018328773964
Squeeze - 5.516017286422173
I Can't Sleep (J'ai pas sommeil) - 5.5096872333124285
Crossfire - 5.49346348095436910

ORACLE

Other Graph Analysis Examples Include …





Community Detection and Evaluation, Link Prediction, Intruder Trace, Network Flow Analysis ……

PGX: Comparison of Features

	Graph Database (Neo4J, Titan, …)	Graph Analytics Framework (GraphX, GraphLab, …)	PGX
Parallel or Distributed Graph Analysis	✗	✓	✓
Pattern Matching (Graph Query)	✓	✗	✓
Transactional Data Management	✓	✗	✓

- Orders of magnitude better performance
- Intuitive DSL and **auto-parallelizing** compiler

- Orders of magnitude better performance
- Expressive graph query language

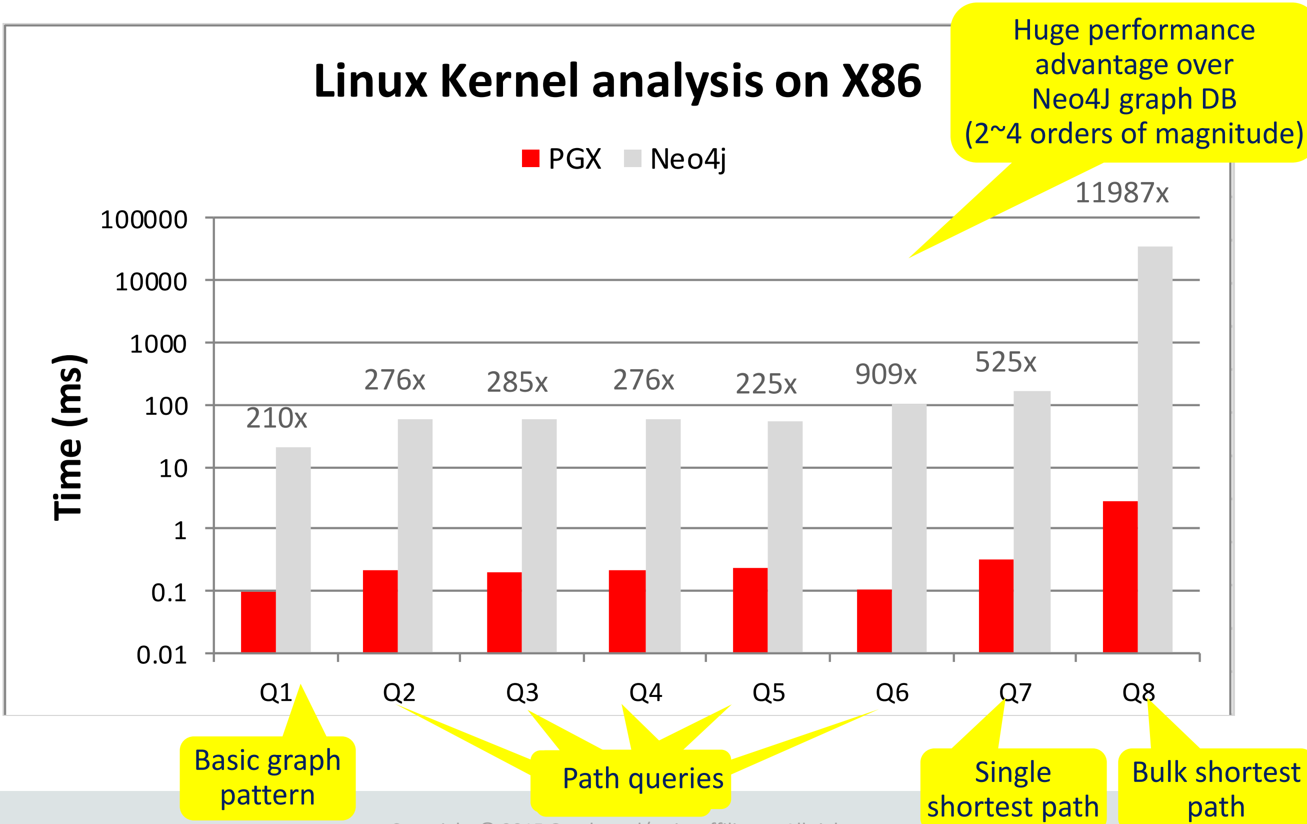
- Integration with proven Oracle Database (RDBMS, NoSQL, …)

ORACLE

PGX: Performance Compared with Neo4J

Path queries of Linux kernel source code

```
X86 Server
Xeon E5-2660 2.2Ghz
 2 socket
 x 8 cores
 x 2HT
256GB DRAM

Neo4J: 2.2.1
Data:
 - Linux kernel code as a graph
 - Program analysis queries
```

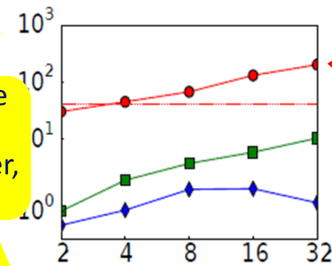


Performance Comparison with GraphX, GraphLab

Benchmarking Result with popular graph algorithms

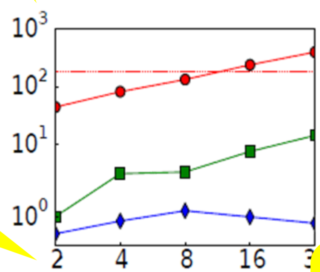
```
Cluster
• Xeon E5-2660,2.2Ghz
  2 socket,8 cores
  2HT, 256GB DRAM)
• 2 ~ 32 Machines
• InfiniBand : 56Gbps

DataSet
TWT: 1.4B edges
WEB: 3.0B edges

GraphX:1.1.0
GraphLab: 2.1
```

Y-axis: Relative Performance (Higher is Better, Log-scale)



WCC(TWT)

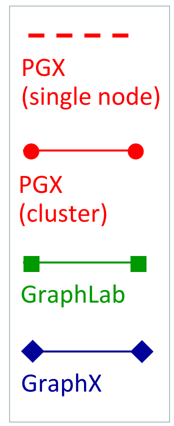
X-axis: Number of machines (2 ~ 32)

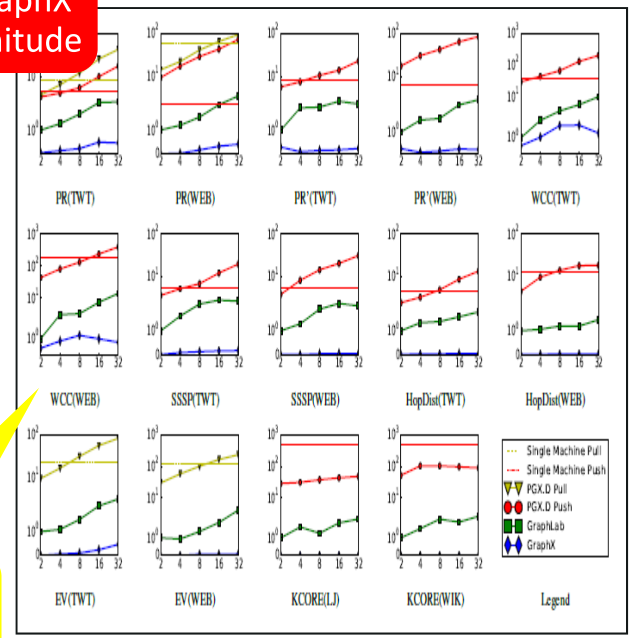


WCC(WEB)

WCC algorithm applied on WEB graph

PGX outperforms GraphLab and GraphX by orders of magnitude

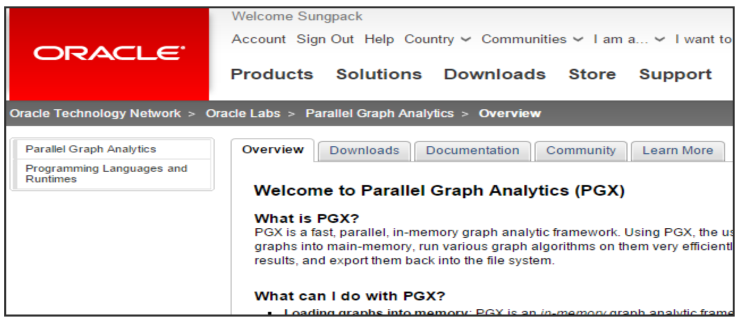




Similar performance gaps across many different algorithms

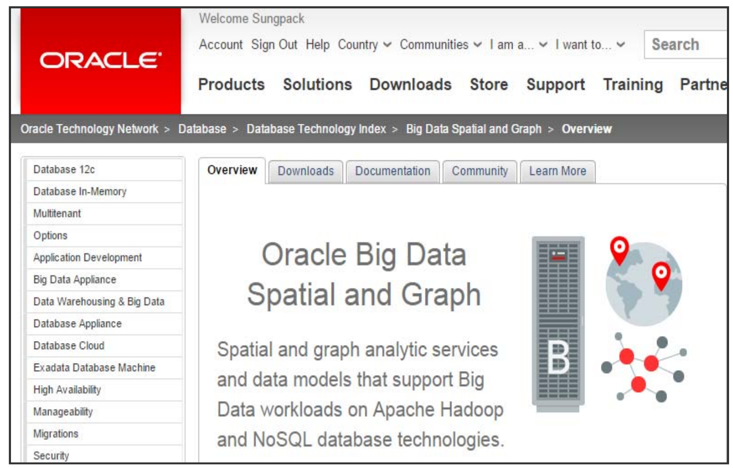
Try PGX Today

OTN Technology Preview

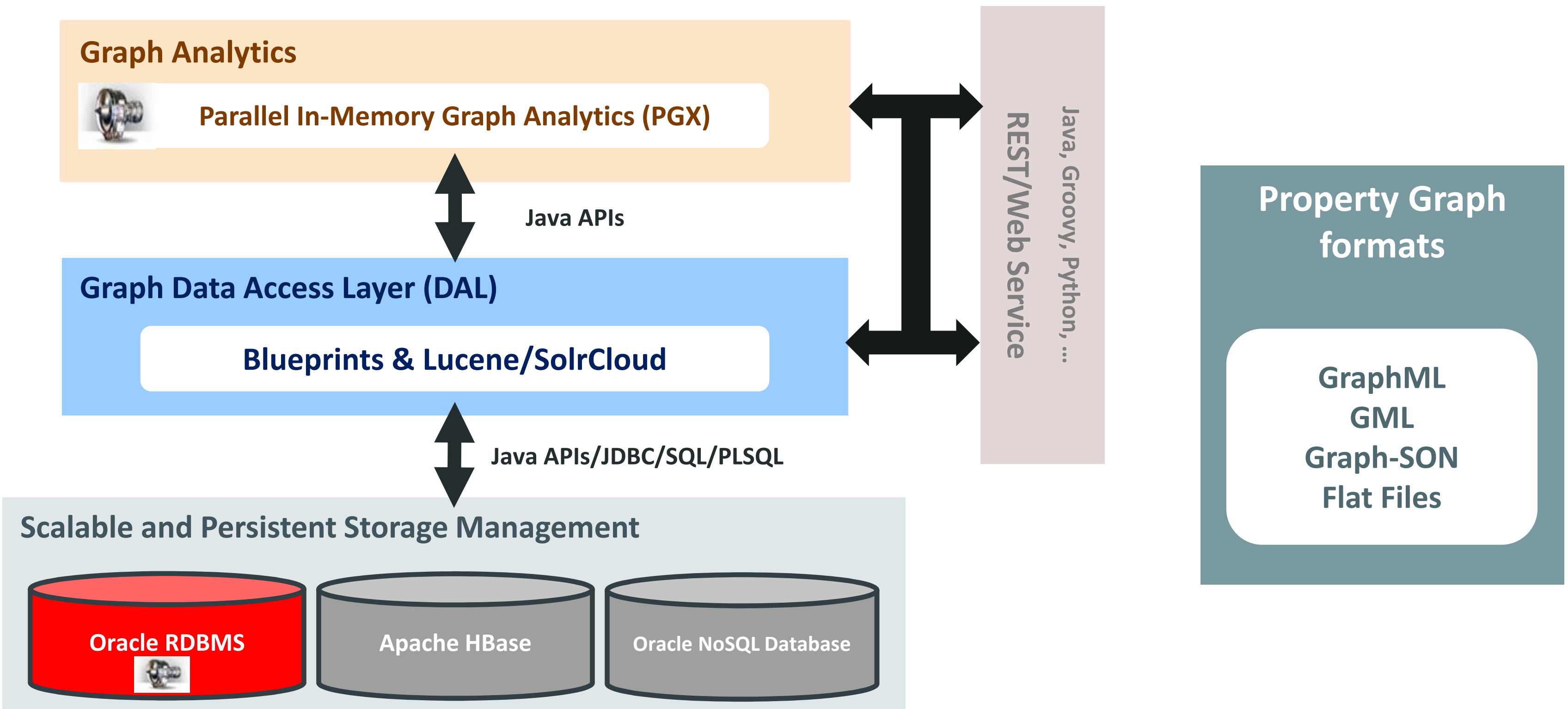


http://bit.ly/pgxdld

Oracle Big Data Spatial & Graph



Architecture of Property Graph Support



ORACLE

Property Graph Database for Oracle Database 12c Release 2

Oracle Spatial and Graph

NEW IN 12.2

- Neo4J/TITAN-like capabilities in Oracle DB
- Massively-Scalable Graph Database
  - Scales securely to **trillions** edges
- In-Memory and In-Database Parallel Graph Analytics
  - **30+ built-in memory-based parallel** graph analysis algorithms
  - **Optimized in-database** scalable graph algorithms
- Simple interfaces
  - **Parallel SQL queries** for graph analysis and filtering of subgraphs
  - Java: Tinkerpop: Blueprints, Gremlin, Rexster
  - Oracle Text, Apache Lucene and SolrCloud
  - Groovy, Python

ORACLE

Conclusion

- Graph Analysis
  - Inspects the fine-grained relationships between data entities
  - Discover implicit information from the data set
  - Many applications
    - reachability and shortest path, influencer identification, recommendation, community detection, …
- PGX is an easy and efficient tool for graph analysis
- Supported component in Big Data Spatial and Graph and Upcoming 12cR2 release of Spatial and Graph Database Option

ORACLE

Hardware and Software
Engineered to Work Together

ORACLE